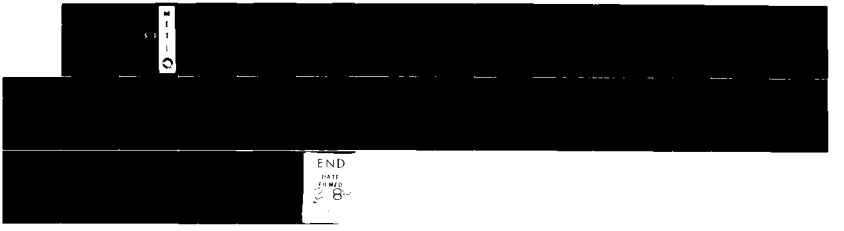
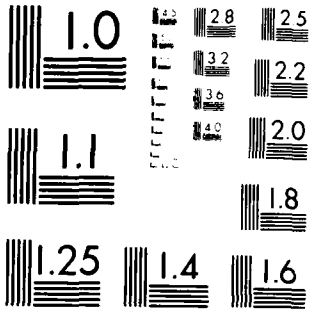
AD-A183 859 A PROGRAMMING ENVIRONMENT FOR PARALLEL VISION ALGORITHMS(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE C BROWN FEB 87 ETL-0457 DACA76-85-C-0001 F/G / 1/1

UNCLASSIFIED ML



END

DATE FILMED 28



ETL-0457

(2)

AD-A183 859

A programming environment for parallel vision algorithms

Christopher Brown

University of Rochester
Computer Science Department
Rochester, New York 14627

February 1987

Prepared for
U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060-5546

E T L

U.S. ARMY ENGINEER TOPOGRAPHIC LABORATORIES
ETL

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30 1986

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S) ETL-0457
6a NAME OF PERFORMING ORGANIZATION University of Rochester 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION U.S. Army Engineer Topographic Labs
6c ADDRESS (City, State, and ZIP Code) Computer Science Department Rochester, New York 14627	7b ADDRESS (City, State, and ZIP Code) Research Institute Fort Belvoir, VA 22060-5546
8a NAME OF FUNDING / SPONSORING ORGANIZATION DARPA 8b OFFICE SYMBOL (If applicable) ISTO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DACA76-85-C-0001
8c ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22209-2308	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62301E PROJECT NO: TASK NO: WORK UNIT ACCESSION NO:
11 TITLE (Include Security Classification) A PROGRAMMING ENVIRONMENT FOR PARALLEL VISION ALGORITHMS	
12 PERSONAL AUTHOR(S) Brown, Christopher	
13a TYPE OF REPORT Annual 13b TIME COVERED FROM 86/2 TO 87/2	14 DATE OF REPORT (Year, Month, Day) 1987 February 15 PAGE COUNT
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02	
17	08	

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Parallel processors
Computer vision
Butterfly computer

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

During the second year of the award period, the Computer Science Department of the University of Rochester continued work in: 1) systems support algorithms, 2) the Butterfly programming environment, and 3) vision applications. This research produced several internal and external reports as well as much exportable code. The University of Rochester also employed DARPA Parallel Architecture Benchmark problems to test different algorithms using four different Butterfly programming environments. These tests produced several interesting results and demonstrated that the Butterfly architecture is a flexible general purpose architecture that can be effectively programmed by non-experts, using tools developed at BBN and Rochester. The University of Rochester is continuing to study the issues and concerns surrounding the effective implementation of parallel algorithms.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Rosalene M. Holecheck	22b TELEPHONE (Include Area Code) (202) 355-2767 22c OFFICE SYMBOL ETL-RI

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Annual Technical Report
DARPA Contract DACA76-85-C-0001
A Programming Environment for Parallel Vision Algorithms
Christopher Brown, P.I.
University of Rochester
11 February 1987

1. Overview

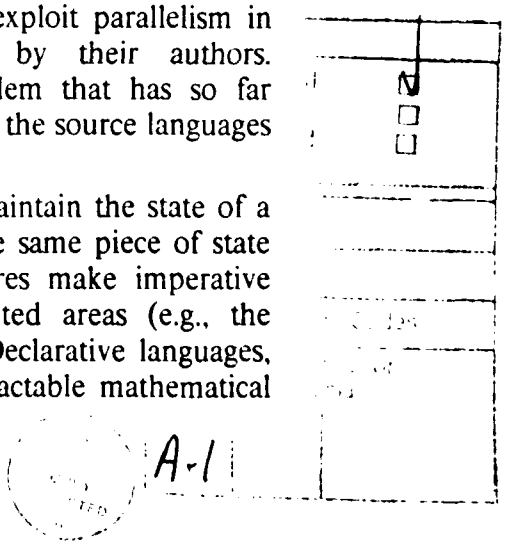
The second year of work in the development of programming environments for parallel vision produced several internal and external reports as well as much exportable code. The reports are detailed in the comprehensive DARPA Contract Reference List at the end of this *Annual Report.* There were appearances at several national parallel computing conferences, with papers on the Butterfly winning various awards. The DARPA Benchmark study and its attendant workshop produced several interesting results. Last, our hardware commitment to real-time vision is increasing with the upgrading of our 16-node butterfly with the BBN-ACI Floating Point Platform package and the acquisition of faster SUN computers and image-processing hardware.

Rob Fowler joined our faculty in September of 1986 and is working in areas relevant to the development of Programming Environments, so his work and plans are reported here also.

2. CONSUL and Related Work

The CONSUL project is an attempt to simplify the use of multi-processor computers for general-purpose programming through automatic detection of parallelism in programs. Current programming techniques for multi-processors require programmers to worry about two related but distinct issues: how to express a solution to their problem as a program, and how to partition this program into parallel pieces. Multi-processor computers will never be as easy to program as sequential ones until programmers are freed from the need to parallelize programs manually. We believe that the best solution to this problem is to develop compilers that will automatically detect and exploit parallelism in programs that have not been explicitly parallelized by their authors. Unfortunately, automatic parallelization is a difficult problem that has so far resisted any general solution. One of the main reasons is that the source languages people are trying to parallelize are inadequate.

Standard imperative languages rely on side-effects to maintain the state of a computation, a problem that is compounded by aliasing (the same piece of state information can have many distinct names). These features make imperative languages impossible to parallelize except in a few limited areas (e.g., the extremely regular code found in scientific computations). Declarative languages, which are generally free of side-effects and have a more tractable mathematical



foundation (important in reasoning about both programs written in them and legal ways of compiling those programs), are more promising starting points for automatic parallelization. Our research is thus focussed on the compilation of *constraint languages* (a particular kind of declarative language) into a form that can be efficiently executed on multi-processors such as the Butterfly.

We have developed a prototype constraint language called CONSUL, which takes its formal basis from set theory. CONSUL and the rationale for its design are discussed in several recent or in-preparation papers: Baldwin and Quiroz, "Parallel Programming and the CONSUL Language", submitted to the 1987 International Conference on Parallel Processing; Baldwin and Quiroz, "Design of the CONSUL Programming Language", circulating internally in draft form, shortly to appear as a technical report; and Baldwin, "Why We Can't Program Multiprocessors the Way We're Trying to Do It Now", in preparation.

The most important on-going work on CONSUL is a set of experiments intended to give us an estimate of the parallelism available from CONSUL programs. Each experiment begins by running a CONSUL program under a crude interpreter --- "crude" because most of the interesting work of solving constraints is done by the user. The interpreter's main purpose is to note when each constraint in a CONSUL program can be satisfied and what variables are defined in the course of doing so. This information is written to a trace file, which is later compacted into a maximally parallel form by a compactor (based on that described in Nicolau and Fisher, "Using an Oracle to Measure Parallelism in Single Instruction Stream Programs", 14th ACM SIGMICRO Microprogramming Workshop, Oct. 1981). Because the traces are taken from actual CONSUL programs in execution, the parallelism found by the compactor is "oracular", i.e., a real compiler could fully exploit it only if it had perfect information about the object program's run-time behavior. Our results will thus indicate the upper bound on the parallelism that can be derived from CONSUL programs. Development of the interpreter began in the Summer of 1986 using Common Lisp on a Sun workstation. In Autumn of 1986 we switched to Texas Instruments Explorer workstations, which provide a more sophisticated Lisp environment for our software. Although moving in-progress work to the new environment delayed us somewhat, the problems have been overcome and work is progressing nicely. The interpreter, although far from complete, is now producing usable traces from simple CONSUL programs. A graduate student (Art Altman) has joined the project and has begun working on the compactor. We expect to have results from the experiments this Summer.

Concurrent with the work on the interpreter and compactor, we have begun exploring a multiprocessor execution model for compiled CONSUL programs. This is a message-passing model, in which processes correspond more or less one-to-one to CONSUL constraints (subject to a number of optimizations that reduce the relative overhead of message passing and process start-up). The messages passed between processes contain (in principle) complete binding environments for variables. This approach eliminates or reduces many problems faced by other

approaches to parallel execution of logic or constraint programs. For example, message passing eliminates the danger of generating incompatible values for variables shared between independent processes, and passing complete environments makes it much easier to synchronize consumers of values to producers than is the case when only single bindings are passed. The price paid for this simplification is that messages are rather large, although there are optimizations that can reduce their size. A complete description of this model appears in "Design of the CONSUL Programming Language" (cited above).

Cesar Quiroz (a graduate student) has been conducting research for his doctorate on automatic parallelization of traditional programming languages since late Summer of 1986. This work is related to the CONSUL project, although not an integral part of it. Cesar's most recent work has been the development of a formal system for reasoning about transformations of sequential programs into parallel ones. This work looks very promising, although it has not yet advanced to the point of actual application.

Work on CONSUL during the next year will emphasize completion of the parallelism experiments, publication of their results, and initial work on a CONSUL compiler (assuming the experiments indicate sufficient parallelism in CONSUL programs). As stated above, we expect the experiments to be finished this Summer. Late Summer and Fall of 1987 should be devoted to writing papers on the experiments, with compiler design beginning in the Winter.

3. Comparing Communication Models

Programming a multiprocessor (or any other parallel computer) involves tradeoffs between parallelism and communication. In order to achieve the best performance, the programmer must maximize parallelism and minimize communication. These goals are in conflict since increased parallelism usually implies increased communication. Although the intrinsic properties of the application limit the amount of parallelism available, communication costs so frequently dominate that the inherent limit to parallelism is rarely reached. One of the few commercially available multiprocessors that does not put a low limit on parallelism and yet still provides efficient communication between processors is the Butterfly.

Conceptually, a tightly-coupled, shared-memory multiprocessor, such as the Butterfly, can support a model of computation based on either shared memory or message passing. The choice of model affects the decomposition of the problem into parallel processes and the resultant granularity of communication. An architecture, such as the Butterfly, that supports both models offers the programmer a wide range of choices for problem decomposition, each with different performance attributes. (The Psyche project, described elsewhere in this report, is an attempt to provide software support for this range of choices.) We performed a series of experiments using Gaussian elimination to evaluate the tradeoffs between shared memory and message passing, over a range of decomposition strategies. Several different implementations were studied and their

performance compared.

One result of these experiments was a set of empirical measures of the effect of decomposition on communication costs. In addition, we have shown that the particular model of computation in use is less important than how well it is matched to the application. The performance of an application depends not only on the efficiency of the underlying communication, but also on the extent to which the underlying model of computation encourages or discourages communication. This work demonstrates the need for a parallel programming environment that supports multiple communication models and decomposition strategies, the focus of our research.

The first results of this work were presented at the 1986 International Conference on Parallel Processing. (This talk received the Distinguished Presentation Award at the conference.) BPR 3 is an early version of that paper. Additional results were presented at the Workshop on Numerical Algorithms for Parallel Computer Architectures sponsored by the Institute for Mathematics and Its Applications and will be published in a Springer-Verlag monograph related to the workshop.

4. SMP and Modula2

The environment of choice for the Butterfly is the Uniform System package from BBN, together with the C programming language. The Uniform System, based on the shared memory model, is the only environment provided by BBN that masks many of the low-level details from the programmer. Therefore, programmers find it easier to use the Uniform System than to build the program from scratch, regardless of how well the application fits the shared memory model. Our experiences have shown the need for support for other models, in particular, message passing.

SMP (Structured Message-Passing) is a simple programming environment for the Butterfly based on message-passing. It is similar in flavor and scope to the Uniform System. SMP provides Butterfly programmers with a model of parallel programs that consists of: (1) *process families*, whose members are created and destroyed together, (2) interprocess communication, within a family, based on *asynchronous message-passing* (send/receive) according to a fixed communication topology, and (3) *a dynamic hierarchy* of such process families. SMP process families and hierarchies add structure to the basic process model of Chrysalis. The advantages of message-passing include increased autonomy for processes, increased efficiency by exploiting locality of data, avoidance of explicit synchronization for data access, and improved protection between processes.

The heavyweight process model of SMP also complements the lightweight process model of Modula-2. We implemented a 68000 code generator for the DECWRL Modula-2 compiler and ported Modula-2 to the Butterfly. Our implementation of Modula-2 provides lightweight processes that share memory (*i.e.*, coroutines), but no true parallelism (ie, the implementation cannot exploit

multiple processors). SMP provides true parallelism via heavyweight processes that communicate using messages, rather than shared memory. Experience has shown that the synergy arising from the combination of lightweight and heavyweight processes is a powerful tool for parallel programming. For many applications, a combination of SMP and Modula-2 could replace the use of C and Chrysalis system calls on the Butterfly.

Our SMP and Modula-2 implementations have been in use on the Butterfly for 6 months. SMP is now being used within BBN and has been requested by several other Butterfly sites. Both software packages will be distributed as part of the BBN user-contributed software release. BPR 4 describes the Modula-2 implementation; BPR 8 describes the SMP system.

5. Chrysalis++

The standard Butterfly programming environment is based on the C programming language and the Chrysalis operating system. This environment requires the user to write a substantial amount of code that relies heavily on the explicit use of pointers, size calculations and type casting, which in turn reduces the amount of compiler type-checking. As a result, programming the Butterfly is extremely error-prone. Since tools for debugging parallel programs are only now being developed (eg, Instant Replay), tracking down these mistakes can be very time-consuming.

Chrysalis++, based on C++, is an effort to improve the situation by using object-oriented programming. C++ provides a richer, and safer, language than C, and Chrysalis++ provides a safe, easy-to-use interface to Chrysalis. The binding strategy for Chrysalis++ is to recast explicit Chrysalis object management into implicit C++ object management. Chrysalis++ merges creation of and access to Chrysalis objects into the declaration of the variable used to represent them. This means that programmers cannot improperly create or access Chrysalis objects. Chrysalis++ also merges object deletion into the automatic C++ variable deletion mechanisms. This approach provides the entire Chrysalis++ programming environment with ne single object management strategy. This strategy is successful in reducing the user code to manage a Chrysalis object from a hundred lines of unchecked C code to one line of strongly-typed C++ code.

We have implemented Chrysalis++ on the Butterfly. Recently, the new C++ compiler from Bell Labs and the new Green Hills C compiler from BBN were incorporated into our environment. The first release is now available for distribution (although legal details concerning C++ licenses must be worked out). BPR 15 describes Chrysalis++ and the Butterfly implementation.

6. Crowd Control

One of the problems with programming a parallel processor is that it is often very difficult to express parallelism, even when the parallelism is obvious. For

example, when a single process broadcasts a message to 100 other processes, the message must be copied from the sender to 100 recipients in sequence. Ideally, it should be easy for the broadcast to be organized in such a way that each process gets the message and then passes it on to several others, thereby performing most of the copy operations in parallel. Since the source and destination processes tend to vary dynamically, this requires the ability to dynamically map tree structures onto a set of processes. Many different problems can make use of this capability, including broadcasting, multiple process creation, and elections.

An important attribute of these problems is that they cannot be executed completely in parallel. For example, all processes cannot copy a message from a single memory in parallel. Some processes must copy the message into their local memory, from which other processes will make copies. Similarly, new processes cannot be created completely in parallel, since some processes must be created by newly created processes. We can implement these problems in parallel by imposing a partial order using an arbitrary embedding in a balanced binary tree. Similar techniques have been used to implement broadcast in local-area networks and internets.

A *crowd* is a set of processes that, at some point in time, cooperate to execute some function in parallel. A partial order is required because some processes make use of previously computed results. Crowd Control is a library package we have developed for the Butterfly that dynamically maps a binary tree control structure onto a set of processes. A partial order is defined by the arbitrary embedding of processes in the binary tree. As a result, N processes can execute a constant time function on N processors in O(log N) time rather than O(N) time. Crowds are dynamic in that processes may join and leave crowds, depending on whether they wish to participate in the execution of crowd functions. Processes that join a crowd can be either homogeneous or heterogeneous.

The time required to serially execute a function F on each of P processors is given by

$$P * (T(F) + C)$$

where T(F) is the time required to execute the function and C is the communication cost. Using crowd control, the same task can be executed by each of P processors in time

$$\lceil \log_2 (P+1) \rceil * (T(F) + C)$$

where C is the communication costs associated with crowd control on a per process basis. In our implementation, C ranges between 2 and 5 ms., depending on the success of certain cache operations. Our empirical data, which ranges over values of P from 1 to 116 and values of T(F) from 1 ms. to 16 ms. is consistent with the analytic results for different values of P and T(F); additional processors

cause the execution time to grow logarithmically. Crowd control is particularly useful when T(F) is greater than the communication costs (2 ms. in our implementation) or P is greater than 16.

The Crowd Control package is now ready for distribution. A BPR describing it is in preparation.

7. Prometheus

The Butterfly architecture provides the illusion of a globally-shared memory. Although any node in the machine may access memory on any other node, there is a substantial performance penalty for doing so, particularly when such references are not amortized via block copy operations. As a result, naming remote memory is a transparent operation (the mechanism used is the same regardless of where the memory resides), but the actual access to remote memory is not (since the performance penalty forces programmers to use block copy operations).

The Uniform System, a library package from BBN that supports the shared-memory model, does not distinguish between naming operations and addressing operations. In other words, a pointer can be used both to access the data to which it refers, and as a generic name for the data to which it refers. Our experience with the Uniform System has shown that the majority of operations on the global address space are naming operations, not access operations. Typical operations on the globally-shared address space consist of passing names (pointers) from one task to another and copying data out of the global address space into local memory. Addresses are used in both cases. The Prometheus project is an attempt to design a shared-memory programming model in which naming and addressing are separate operations. The three salient features of Prometheus are:

- lightweight tasks
- a global name space
- local data spaces

Prometheus uses an addressing scheme characterized by a global name space and a local data space. The name space is accessible to all tasks running in the system and every task has its own local data space. We refer to the global name space as the Qspace.

The Qspace memory makes a clear separation between global and local memory. Once a datum has been placed in the global memory, the operations that can be performed on it are extremely limited. In order to actually change the value of a global datum, it must be removed from the Qspace, the appropriate value changed, and then reinserted into the Qspace. The act of removal moves the datum from global memory to local memory. Hence, all data manipulation is done on local memory. All name manipulation is done on global memory and the operations on that memory are rigidly defined. The organization of the Qspace is based loosely on the Linda distributed programming language, although the Qspace is much simpler than the tuple-space of Linda.

We have recently finished the first iteration of the Prometheus design. A BPR describing the design of the Qspace is in preparation.

8. Elmwood

Last semester, as a learning exercise, a graduate class designed and implemented an object-based operating system for the Butterfly called Elmwood. This project was the first attempt outside of BBN to build an operating system for the Butterfly. In Elmwood, objects are passive entities. Processes communicate using remote procedure calls to objects. Object handles are kernel-protected, so an object has full control over the set of processes that may manipulate it. A process's address space appears to grow and shrink dynamically as a result of procedure calls to objects.

Although Elmwood was undertaken as a class project, it has made concrete contributions to our research effort. First, it has significantly expanded our understanding of the internal workings of the Butterfly. Second, we have developed sufficient expertise and software to allow us to consider constructing an operating system (such as Psyche) without using Chrysalis as a base. Third, many of the ideas in Elmwood are being explore as part of the Psyche project. A BPR describing Elmwood and its implementation is in preparation.

9. Instant Replay

In order to understand an execution of a program, we must be able to characterize it in a form amenable to study and analysis. The output produced when the program is executed is one possible characterization, although it is usually too coarse to enable us to understand subtle aspects of the program's execution. For sequential programs, the code and input data offer a static, fine-grain characterization of an execution, since we can use them to produce the execution and then examine it at an arbitrary level of detail. This particular fine-grain characterization does not suffice for parallel programs. Different executions of the same parallel program with the same input often do not produce the same results due to race conditions. No static characterization can capture such behavior.

We have developed a dynamic, fine-grain characterization of parallel program execution based on a partial order of accesses to shared objects. To capture a description of a particular execution of a program, we record the relative order of interprocess events as they occur during execution. Such a record of accesses is expected to be much smaller than a record of all data exchanged between processes, thereby making this description of an execution relatively easy to capture. Once we have recorded an execution description, it can be used by a variety of tools to help the programmer understand the behavior of the program. One such tool, a prototype called Instant Replay, has been implemented on the Butterfly. Instant Replay uses an execution description to reexecute a parallel program, enforcing repetitive behavior otherwise precluded by race conditions. The deterministic behavior offered by Instant Replay makes the standard

debugging cycle possible for parallel programs.

Recently, we have modified the monitoring protocol and improved its performance. Monitoring now affects the overall performance of our test program, Gaussian elimination, by only about 1%. We are also working on a version of the protocols for loosely-coupled systems. A paper describing this work, "Debugging Parallel Programs with Instant Replay," has been accepted for publication in IEEE Transactions on Computers and will appear in April 1987. BPR 12 is an early version of that paper. In addition, a position paper on our basic approach was presented at the IEEE Workshop on Instrumentation for Distributed Computing Systems.

10. LYNX

LYNX is a high-level programming language for distributed computing. It was originally designed at the University of Wisconsin - Madison, where it was implemented on the Crystal multicomputer. We began a port to the Butterfly in September of 1985. A partial implementation became available in early 1986, and a full version (including a number of features not provided at Wisconsin) was finished by the middle of the summer.

In the context of our work on the Butterfly, LYNX serves both as a language design research project and as a day-to-day programming environment for applications development. As a language design project, its goal is to support both application and system programs in a single conceptual framework. Unlike other distributed languages, LYNX relies on efficient run-time binding and checking to allow processes to be compiled independently, without knowledge of their peers. A newly-created process is therefore able to communicate, within the language, with separately-developed processes already in operation. The programmer can not only run an application composed of many cooperating processes, but can also create server programs that remain in operation, providing their services to each application in turn. A novel coroutine-like mechanism supports the automatic management of context within a server for interleaved conversations with an arbitrary number of clients.

In its role as an applications testbed, LYNX provides a significantly higher level of convenience, functionality, and type security than is possible with a library package for interprocess communication. Since LYNX is a message-based language, processes do not in general share memory (though they can arrange to do so through use of one of the standard LYNX libraries). Communication normally takes place over two-way communication channels called **links**. The ends of links can be passed back in forth in messages, permitting dynamic reconfiguration of the process interconnection graph. Communication statements refer to arbitrary collections of program variables, which are gathered and scattered automatically. Strict type checking is enforced. Errors in communication result in high-level language **exceptions**, similar to those of Ada. The integration of message passing with the coroutine facility combines the conceptual advantages

of blocking remote procedure calls with the performance advantages of non-blocking, buffered messages.

10.1. Butterfly Implementation

Our implementation of LYNX has been in steady use since late summer. The final features to be added included integration of the exception-handling mechanism with the catch/throw facilities of Chrysalis, and the use of pointers to provide shared access to memory objects by multiple processes. A new utility routine was developed this fall that allows one process to raise an exception in another, allowing the other to perform cleanup operations before terminating. This routine has been incorporated in a stand-alone tool program that can be used by anyone on the Butterfly. It is of particular help to users of LYNX, since it can be used while debugging to terminate a program in an infinite loop in such a way that exceptions propagate to all communicating processes.

The largest application completed to date is a checkers-playing program based on Fishburn's algorithms for parallel alpha-beta search. Experimentation with this program continues, and is expected to result in a case study paper sometime this next year. LYNX also served as the implementation environment for a programming assignment in one of our graduate courses this past fall, and for one of the applications in the DARPA parallel algorithms benchmark study in August.

A complete language reference manual was published as Rochester BPR #7. A comparison of three implementations of LYNX was presented at the 1986 International Conference on Parallel Processing in August. A paper on the language design and rationale appeared in the January 1987 issue of *IEEE Transactions on Software Engineering*. A technical note on the LYNX type-checking mechanism is still in publication delay at the same journal.

10.2. Performance Analysis

Experience with a wide range of multicomputer and multiprocessor systems software suggests that message passing is often three or four orders of magnitude more expensive than shared memory for communication between parallel processes. Differences in the speed of underlying hardware mechanisms fail to account for a substantial portion of the performance gap. The remainder is generally attributed to the "inevitable cost" of higher-level semantics, but a deeper understanding of the factors that contribute to message-passing overhead has not been forthcoming. Though message-passing systems address such issues as synchronization, buffering, flow control, authentication, address resolution, type checking, exception handling, and error correction, none of which is provided with shared memory, there remains a widespread belief among researchers in parallel systems that current message-passing facilities are somehow slower than they ought to be.

In an attempt to address the issue of "where the time goes" in message passing systems, we undertook a detailed study of the Butterfly implementation of

LYNX. Our study involved a number of preliminary measurements, a series of protocol optimizations designed to improve performance, and a detailed analysis of remaining costs. The data provide a direct measure of the expense of individual features in LYNX. They also provide insight into the likely costs of other message-passing systems, both present and future.

Careful attribution of costs to message-passing subtasks is the only reliable way to evaluate the cost-effectiveness of contemplated features. Without detailed accounting, it is impossible to determine whether the difference in speed between competing systems is due to hardware overhead, choice of semantics, or simply cleverness of coding. We believe LYNX to be representative of a large class of languages in which interprocess communication is based on rendezvous or remote procedure call. Many of our results should generalize directly. In addition, our experience in measurement techniques should be of use to other researchers in performing similar studies.

The results of our study are reported in Rochester BPR #17, which has been submitted to the 7th International Conference on Distributed Computing Systems. In addition to apportioning overhead among some twenty different functions, the paper provides a timeline that explains what is happening during each microsecond of a remote invocation. As a result of our protocol enhancements, trivial remote operations in LYNX new complete in about 1.8 ms, a time that is competitive with the fastest comparable systems. We are eager to see the effect of the 68020 processor upgrade on our figures.

11. Network Support

As software is developed to make multiprocessors more useful, it becomes increasingly important to have high-quality graphics interfaces with which to interact with these machines. In an environment such as ours, where users routinely use the multiprocessor remotely from powerful personal workstations with bit-mapped displays, it also becomes important for user interface and other software to work across the network. These concerns have been addressed at BBN by providing Butterfly support for the X windowing system. They have also prompted the development here at Rochester of a remote command daemon compatible with the Berkeley UNIX **rsh** facility.

X is an increasingly popular, network-transparent windowing system developed at MIT. Implementations of X are available on a wide variety of machines. The Butterfly implementation was constructed by one of our own graduate students, Ken Yap, during a summer internship at BBN. This fall, as part of a research assistantship, Ken installed the Butterfly X tools in our local environment and developed an interface for X that allows it to be used from LYNX. Since the X library contains several hundred interface routines, with a wide variety of parameter and data structure types, the interface description files provided an excellent opportunity to test the LYNX compiler.

The X interface has been used to write a LYNX version of the standard "plaid" program, a graphics demonstration that displays a shifting pattern of diagonal bars created by flipping pixels along an interfering pair of moving borders. It has also been used to construct a mouse-based graphical interface to the LYNX checkers-playing program.

To allow Butterfly programs to be executed remotely from our UNIX machines, without logging in directly, we also developed a daemon process that runs on the multiprocessor and supports the Berkeley UNIX **rsh** protocol. A user at a workstation, for example, can now type a single command that causes the daemon to create a new shell (on the Butterfly) which in turn allocates an appropriate number of Butterfly nodes (see the section on Software Partitioning, elsewhere in this report) and runs the checkers program. Using the X library, the checkers program then creates a new graphics window back on the workstation's screen in which to play the game. No explicit log-on to the Butterfly is required.

To facilitate use of the **rsh** daemon, we implemented enhancements to the Butterfly shell program that 1) allow a newly-created shell to be passed an initial command, and 2) allow multiple commands to be specified on a single line. These facilities are standard in the UNIX shells. Our modifications of the Butterfly shell have since been adopted by BBN.

12. Software Partitioning

One of the most serious limitations of the Chrysalis operating system has always been its assumption that the machine belongs to a single, current user. Independent applications have been unable to coexist on the same machine, partly because of protection problems, and partly because there has been no mechanism for partitioning certain fundamental system resources.

With the dramatic increase in use of the Butterfly through the summer of 1986, it became apparent that support for multiple users was essential in our local environment. Our professional staff undertook to develop software that allows the nodes of the Butterfly to be partitioned into virtual machines. Users logging into a Butterfly now receive a single node by default, and can arrange to acquire or return additional nodes dynamically. Protection problems remain (and in fact cannot be solved completely with the current hardware architecture), but accidental interference between programs has been made relatively unlikely.

At the fall Butterfly Users Group meeting, we learned that BBN had been pursuing partitioning independently. Our groups have since been in contact to ensure that the best parts of both approaches make their way into the official software release.

13. Modula-2 Event Package

Certain kinds of parallel algorithms, particularly graph algorithms, are most easily coded with extremely large numbers of processes (on the order of thousands or tens of thousands). Until recently, no programming environment on the

Butterfly currently supports such applications. We were made acutely aware of the limitation during the DARPA benchmark study, when we would have liked to code graph algorithms with one process per node of the graph. Though Modula-2 and LYNX both provide lightweight threads (coroutines) inside Chrysalis processes, the mechanisms used to communicate between threads in the same process are completely different from the mechanisms used to communicate between threads in different processes.

In order to provide *uniform* communication between very large numbers of threads, we undertook this past fall to implement a library package in Modula-2 that encapsulate the blocking Chrysalis operations, allowing them to be called from Modula-2 coroutines without blocking an entire process. An operation (such as waiting for an event) that would normally block the process now places the current coroutine in a library data structure and switches to another coroutine instead. When there are no other runnable coroutines in the current process, the library waits for an event and unblocks the appropriate coroutine.

The basic mechanism is similar in flavor to the thread facility in LYNX. The difference is that the events library ties the blocking and unblocking of threads to the fundamental scheduling mechanism of the Butterfly, rather than to a particular style of interprocess message passing. Though the events package lacks all the LYNX advantages of high-level language convenience, it can be used to implement any library package-based communication facility that is feasible on the Butterfly. We hope eventually to extend the package with several higher-level communication facilities, perhaps in conjunction with the Psyche project described below.

14. Psyche

Conventional wisdom holds that parallel processes must communicate either by sharing memory or by exchanging messages. These alternatives are generally viewed as incompatible opposites. It is our contention, however, that conventional approaches are better regarded as points on a *continuum* that reflects the *degree of sharing* between processes. The full spectrum includes many different styles of message passing, as well as monitors, path expressions, remote procedure calls, atomic data structures, and unconstrained shared memory. In a pure shared-memory approach, processes share everything; in a pure message-passing approach, they share nothing. The other options lie somewhere in-between.

The Psyche project is an attempt to construct a general-purpose multiprocessor operating system. By general-purpose we mean 1) that the operating system will run almost any application for which the hardware is appropriate, and usually run it well, and 2) that it will support both individual, highly-parallel applications and larger numbers of users with smaller applications, in the style of conventional time sharing. We consider the support of large-scale parallelism to include the ability to choose the communication model most appropriate for each individual application, or *piece* of an application.

The fundamental abstraction provided by the Psyche kernel is the **realm**. A realm is essentially an abstract data object; it consists of code, data, and references to other objects. The data can be accessed *only* through invocation of the code. Through a system of kernel-protected capabilities, incremental changes to address spaces, and direct execution of remote operations, Psyche will address the conflicting goals of efficiency, flexibility, and security. Realms will permit the implementation of a wide variety of process and communication models. They will allow multiple models to coexist on a single machine, and even within a single application.

Our work so far has focussed on the abstractions provided by the Psyche kernel. We have worked through several iterations of the kernel interface specification and are now assembling a design document to be submitted for publication. We hope to be in a position to begin implementation sometime this spring. Eventually, we plan to explore the implications of Psyche primitives on programming language design. Our goal is to investigate the extent to which multiple models of interprocess communication can be supported within a common language.

15. Motion and Real-time AI

During 1986, Aloimonos' work centered on the robust and reliable computation of intrinsic images, or physical parameters of the scene. He invented several new techniques, and his method has been to add information sources rather than to rely exclusively on apriori constraints (such as smoothness). His work has mainly been in the domains of multiple frame vision (stereo, motion) and in texture. Bandopadhay was also working in the domain of motion. His work has been to apply clustering to the motion segmentation and egomotion problem, and to notice that proprioceptive feedback from tracking stationary points can work with vision to make the egomotion calculations easier.

The tracking work is the scientific motivation for certain robotic hardware designed and built at Rochester, which consists of two cameras on a "robot head". With this setup we are investigating real-time vision. A pilot project by Dave Coombs and Brian Marsh has produced a software framework and a working program that tracks multiple moving colored objects in a scene. This work emphasizes multi-resolution processing, focus of attention, and real-time and priority job scheduling, but does not mechanically move the cameras to implement its active attention control. Tom Olson is continuing this work. With the acquisition of new and faster host computers, new and faster low-level image analysis hardware and hardware upgrades for our Butterfly Parallel Processor, we hope to build a multi-camera system that can track and analyse multiple objects in real time in navigational and robotic contexts.

15.1. BIFF: A Butterfly Vision Library

Tom Olson and Liud Bukys have constructed a parallel version of the IFF vision library written at the University of British Columbia under the direction of

Prof. Havens. IFF is a file organization for images, and an associated set of image processing and vision utilities, something like SPIDER or GIPSY. IFF programs are written as UNIX filters, and the system uses UNIX pipes to concatenate operations. This is a slow way to go about things but is very modular and good for interactive use. BIFF, the parallel version, is much faster, both through capitalizing on the innately parallel nature of many low-level vision operations, and through use of the large memory on the individual butterfly nodes to achieve "in-core" files that can be passed from process to process quickly through memory mapping. Tim Becker has extended BIFF to contain a version of Burt's gaussian-filter resolution pyramid. We expect BIFF to be a useful tool and to expand in the future.

15.2. Segmentation with the Uniform System

Tom Olson has constructed an advanced program under the Uniform System to do segmentation. We are interested in the general problem of combining the outputs of low-level vision processes to produce robust interpretations of large classes of input images. In addition, we want solutions that make efficient use of large-scale parallel hardware. In order to study these issues we have chosen a particular well-studied problem (2-d segmentation) for implementation on the BBN Butterfly Multiprocessor. To date we have been more concerned with communications and systems aspects of the combination than with the mathematical aspects of cooperating constraints or evidence combination.

The program works by recursively splitting regions until all regions satisfy some termination criteria. Users of the system must provide a set of functions called *experts* which take as their argument some region and generate a proposed segmentation of that region. The user also provides a *reconciling* function which integrates a set of proposed segmentations into a single proposal, which the main program then executes. With minor changes a large class of currently used segmentation algorithms can be fit into this model. Among its defects are that a) there is no provision for merging and b) reasoning based on more than one region (eg based on connectivity) is forbidden. The current implementation has only one expert function, a grey-level histogram splitter loosely based on PHOENIX, the multispectral segmentor of Shafer and Kanade[1]. The program is well parallelized, using the Uniform System library to implement parallel loops. Users provide an initialized vector of pointers to the expert functions and the reconciler. The program makes use of BIFF (see above).

This segmentation program illuminated many issues of parallelization, load balancing, and models of computation for tightly coupled MIMD machines such as the Butterfly.

16. Multi-modal Segmentation

Paul Chou is building a segmentation program that uses multiple modes of information (intrinsic images such as depth and local surface orientation, structured light, image intensity). His approach uses Markov Random Fields as a

way of expressing the probability of local configurations of evidence. A method of combining likelihoods is used to do incremental evidence combination. So far the likelihoods have been provided by Dave Sher's operators (see below). Local evidence combination yields intermediate results that are combined by more global methods (for instance grouping processes and methods to fit parameterized surface models to data) into segments (descriptions of three-dimensional surface patches).

17. Parallel Vision Algorithms

The Darpa Benchmark [Brown et al 1986] problems were used to exercise several newly-developed programming libraries and languages. In three weeks, each benchmark problem was implemented, usually in several versions, by programmers varying in skill from Butterfly experts to complete Butterfly beginners. Our goal was primarily to test different algorithms and also sometimes to try different programming environments on the Butterfly. The Benchmark work resulted in four Butterfly Project Reports (BPRs 10, 12, 13, and 14), available from Rochester's CS Department.

Four programming environments (two developed at Rochester) were used: C and Chrysalis (the Butterfly operating system), the Uniform System (a shared memory model by BBN, with local improvements), LYNX (a systems language with communication links as primitive objects developed by Michael Scott), and Structured Message Passing (a message-passing and process family model developed by Tom LeBlanc).

We conclude that the Butterfly architecture is a flexible general-purpose one (not a peripheral vision processor). It can be effectively programmed by non-experts, using tools developed at BBN and Rochester. Our work underlined several general points. There are conceptual and practical tradeoffs in acknowledging or disguising the existence of local memory. Serializations hidden in the operating system can affect performance significantly. Some enhancements to the microcode would be very helpful. A faster processor would help for certain applications, and might help reveal the next system bottleneck. More detailed conclusions are available in BPR-13.

Four Butterfly Project Reports were generated in the Benchmark exercise. The results were also briefly reported by Azriel Rosenfeld at the DARPA IU Workshop at USC in February 1987. The final timings were as follows.

Convolution: 3.48
Zero-Crossings: .16
ChainCodes: 1.47
Hough Transform (10000 points): 1.2
Convex Hull: .17
Visibility: 2.5
Subgraph Isomorphism: 2 solns/sec.
Min Cost Path (all pairs): Linear Speedup to 20 processors.

18. Effective Implementation of Parallel Algorithms.

The central motive for deciding to use a parallel machine for the solution of some problem is to achieve a much faster solution than is possible on sequential machines. If we do not achieve a large speedup the expense of the parallel machine and the software that runs on it will have been wasted. It is therefore imperative that we achieve all the speedup we can.

The problem of going from a specification of a problem to a running program that solves the problem involves several levels of translation and transformation:

> Given the specification we design an abstract algorithm that is judged to be efficient with respect to some abstract model of computation.
>
> The abstract algorithm is programmed in an appropriate language supported by a programming system.
>
> The programming system translates this into something that runs on the underlying physical hardware.

Our success in achieving good performance depends upon all three of these transformations. Each level of abstraction must by itself embody the requisite degree of parallelism and each transformation must be efficiently mapped onto the next. A failure anywhere in this chain can destroy the efficacy of the entire process.

We are therefore studying this process, attempting to evaluate our success in each of these translation processes. Questions of interest include the appropriateness of the abstract models in which the algorithms are designed, the kind of abstract model that the programming system presents to the programmer, and the success with which the programming system makes effective use of the underlying hardware.

In order to examine this process in more detail we are taking a two-pronged approach.

On one hand, we are examining theoretically interesting parallel algorithms and are beginning to produce multiple implementations of them in the various programming environments available on the Butterfly. This began as part of the "DARPA Parallel Architectures Benchmark Study" [BPR 13]. The emphasis is upon problems that appear often as subroutines in larger computations. Included are set operations, manipulations of graphs,sorting, and parallel prefix computations [M. Fischer and R. Ladner, "Parallel Prefix Computation", JACM 27,4 p831. and C. Kruskal, L. Rudolph, and M. Snir, "The Power of Parallel Prefix", IEEE Trans. on Computers C-34,10, Oct.85, p965.]

A difficulty that arose in the Benchmark study was explaining why some implementations of highly parallel algorithms initially exhibited disapointingly poor speedup. In almost all cases such poor performance is a manifestation of Amdahl's law; there is some part of the implementation that is sequential and this serves as a bottleneck limiting overall performance. Possible sources include sequential sub-computations in the abstract algorithm, serialization introduced by

the programmmer of the algorithm, serialization introduced by the programming system, or sequential operations provided by the underlying operating system and hardware architecture.

It is often difficult to discern which of these is to blame and the programmer is left to his own intuition as to the source of the problem and how to correct it. The second thrust of our efforts is to construct performance monitoring and analysis tools that can be used systematically to examine these issues.

19. Performance Analysis Tools

The ultimate criterion for evaluating a practical parallel algorithm is how well an implementation of it runs on a physical machine. Analyses of the higher levels of abstraction are useless if the implementations of those levels introduce hidden costs, either because the abstract model does not adequately represent the true costs of the operations it provides, or because the system introduces hidden sequentialization.

To obtain both quantitative and qualitative estimations of this we are in the process of implementing parallel performance monitoring tools. The goal is to provide a programmer with access to detailed information about the performance of his or her program.

The simplest form of performance monitoring that we have is a modified version of the Uniform System that gives the programmer feedback as to the status of each of the processors used by the program. As each processor is queried approximatelyu four times a second, this method is useful for obtaining only very coarse information on processor utilization.

The second form of performance monitoring uses the program counter profiling facility provided by Chrysalis. This can be used to estimate the utilization of each processor and which code it spends most of its time executing. Utilities to facilitate its use are currently under construction. Unfortunately, it is difficult to tell from the information collected by the profiler why a particular processor is being poorly utilized.

The most promising approach to performance monitoring and analysis that we are pursuing is the extension of the Instant Replay debugging system to provide enough information to do an off-line performance analysis. As described above, Instant Replay facilitates the debugging of parallel programs by the use of "history tapes" that record synchronization events among processes. The histories are a means of deterministically repeating a particular execution with respect to those events.

For performance monitoring and analysis purposes the events recorded in a history are being extended to include the time at which a process arrives at a synchronization point and, if it is forced to wait, the time at which it is able to resume execution. Once recorded, the history will be uploaded to a workstation for analysis. In addition to gross statistics on processor utilization, the interactive

analysis tools will be able to detailed data on the interactions among individual processors, thus revealing botttlenecks and other dependencies at which a nominally parallel computation is serialized.

The core of the initial test suite of programs to be analyzed consists of the programs written for the DARPA Parallel Architecture Benchmark Study. In addition we are using a version Batcher's odd-even parallel sort [Knuth *Art of Computer Programming* Vol. 3, p.225] using a "merge-split" step at each internal step of the process rather than just a compare and swap. We have this running under Chrysalis and will also code up versions to run under our other programming systems. We will use the Chrysialis implementation to investigate the sensitivity of the program to process to processor allocation due to switch and memory contention. In addition it will serve as a basis for evaluating the other systems.

Publications Reporting Research Sponsored in part by DARPA/ETL Grant No. DACA76-85-C-0001 and in part by DARPA ONR Grant No. N00014-82-K-0193

1985 - - 1986

Allen, J.F., "Maintaining Knowledge About Temporal Intervals," in *Readings in Knowledge Representation*. R. Brachman and H. Levesque (eds). Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1985.

Allen, J.A., "Natural language lecture notes," Computer Science Dept., Univ. Rochester, January 1986; to appear, contracted book, Benjamin Cummings Publishing Company.

Allen, J.F., "Speech Acts," in *Encyclopedia of Artificial Intelligence*. S. Shapiro (ed). New York: John Wiley & Sons, Inc., in press, 1987.

Allen, J.F. and P.J. Hayes, "A common-sense theory of time," *Proc., 9th Int'l. Joint Conf. on Artificial Intelligence*, Los Angeles, CA, August 1985; TR 180 (longer version), Computer Science Dept., Univ. Rochester, to appear, 1986.

Allen, J.F. and H.A. Kautz, "A Model of Naive Temporal Reasoning," in *Formal Theories of the Common Sense World (Vol. 1)*. J.R. Hobbs and R. Moore (eds). Norwood, NJ: Ablex Publishing Co., 1985.

Allen, J.F. and D.J. Litman, "A plan recognition model for subdialogues in conversations," to appear, *Cognitive Science*, 1987.

Allen, J.F. and D.J. Litman, "Plans, goals and natural language," *1986-87 Computer Science and Engineering Research Review*, Computer Science Dept., Univ. Rochester, October 1986; *IEEE Special Issue on Natural Language Processing* 74, No. 7, July, 1986, 939-947.

Allen, J.F. and R. Pelavin, "A formal logic of plans in temporally rich domains," *IEEE Special Issue on Knowledge Representation*, 74, 10, October 1986.

Allen, J.F. and C.R. Perrault, "Analyzing Intention in Utterances," in *Readings in Natural Language Processing*. B. Grosz, B. Webber, and K. Sparck-Jones (eds). Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1986.

Aloimonos, J., "Computing intrinsic images," Ph.D. thesis and TR198, Computer Science Dept., Univ. Rochester, September 1986.

Aloimonos, J., "Shape and motion from contour, without point to point correspondence: general principles," TR 173, Computer Science Dept., Univ.

Rochester, to appear, 1986; *Proc., IEEE Computer Vision and Pattern Recognition*, Miami, FL, June 1986.

Aloimonos, J., "Structure from motion: I) optic flow vs. discrete displacements; and II) Lower bound results," *Proc., IEEE Computer Vision and Pattern Recognition*, Miami, FL, June 1986.

Aloimonos, J. and A. Bandopadhay, "Perception of structure from motion: Lower bound results," TR 158, Computer Science Dept., Univ. Rochester, March 1985.

Aloimonos, J., A. Bandopadhay, and P. Chou, "On the foundations of trinocular machine vision," *Technical Digest, Topical Meeting of the Optical Society of America*, Lake Tahoe, April 1985.

Aloimonos, J. and A. Basu, "Determining the translation of a rigidly moving surface, with correspondence," TR 176, Computer Science Dept., Univ. Rochester, to appear, 1986; *Proc., IEEE Computer Vision and Pattern Recognition Conf.*, Miami, FL, June 1986.

Aloimonos, J., A. Basu, and C.M. Brown, "Coutour, shape and motion," *Proc., DARPA Image Understanding Workshop*, Miami, FL, December 1985.

Aloimonos, J. and C.M. Brown, "Perception of structure from motion, 1) Optical flow vs. discrete displacements, II) Lower bound results," *IEEE Conf. on Computer Vision and Pattern Recognition*, Miami Beach, FL, June 1986.

Aloimonos, J. and C.M. Brown, "Robust computation of intrinsic images from multiple cues," in *Advances in Computer Vision*, C. Brown (ed.), Lawrence Erlbaum, in press, 1988.

Aloimonos, J. and P. Chou, "Detection of surface orientation and motion from texture I: The case of planes," TR 161, Computer Science Dept., Univ. Rochester, January 1985.

Aloimonos, J. and P. Chou, "Detection of surface orientation from texture," *Optic News*, September 1985.

Aloimonos, J. and I. Rigoutsos, "Detection of 3-D motion without correspondence: I) Planar surfaces: theory and experiments; II) Curved surfaces, theory," TR 178, Computer Science Dept., Univ. Rochester, December 1985.

Aloimonos, J. and I. Rigoutsos, "Determining the 3-D motion of a rigid planar patch without correspondence, under perspective projection," *Proc., Canadian Artificial Intelligence Conf.*, Montreal, 1986; also *Proc., IEEE Workshop on Motion*, Charleston, SC. May 1986.

Aloimonos, J. and I. Rigoutsos, "Determining 3-D motion of rigid surfaces without correspondence," *Proc., AAAI 1986*, Philadelphia, PA, August 1986.

Aloimonos, J. and M.J. Swain, "Shape from texture," *Proc., 9th Int. Joint Conf. on Artificial Intelligence*, Los Angeles, CA, 926-931, August 1985.

Baldwin, D. "A model for automatic design of digital circuits," TR188, Computer Science Dept., Univ. Rochester, July 1986.

Baldwin, D. "AI, algorithms, and hybrids for electronic design," submitted, Applications of Artificial Intelligence in Engineering, Conference, 1986.

Baldwin, D. "Physical constraints in behavioral synthesis," submitted, 24th Design Automation Conf., 1986.

Baldwin, D., "Parallel programming and the Consul language," submitted, International Conf. on Parallel Processing, 1986.

Bandopadhay, A., "Constraints on the computation of rigid motion parameters from retinal displacements," TR 168, Computer Science Dept., Univ. Rochester, October 1985.

Bandopadhay, A., "A computational study of rigid motion perception," Ph.D. thesis, Computer Science Dept., Univ. Rochester, September 1986.

Bandopadhay, A., "Perception of structure and motion of rigid objects," TR 169, Computer Science Dept., Univ. Rochester, December 1985.

Bandopadhay, A. and J. Aloimonos, "Perception of motion of rigid objects," TR 169, Computer Science Dept., Univ. Rochester, December 1985.

Bandopadhay, A. and J. Aloimonos, "Perception of rigid motion from spatiotemporal derivatives of optical flow," TR 157, Computer Science Dept., Univ. Rochester, March 1985.

Bandopadhay, A., and D.H. Ballard, "Visual navigation by tracking of environmental points," *SPIE Conf. on Artificial Intelligence*, Orlando, FL, March 1986.

Bandopadhay, A., B. Chandra, and D.H. Ballard, "Egomotion perception using active vision," *Proc., IEEE Conf. on Computer Vision Representation and Control*, Miami Beach, FL, June 1986.

Bandopadhay, A., B. Chandra, and D.H. Ballard, "Active navigation: tracking an environmental point considered beneficial," *IEEE Workshop in Motion Representation and Analysis*, Charleston, SC, May 1986.

Bandopadhay, A. and R. Dutta, "Measuring image motion in dynamic images," *IEEE Workshop on Motion, Representation and Analysis*, Charleston, SC, May 1986.

Bandopadhay A. and R. Dutta, "Measuring notion in dynamic images: a clustering approach," *6th Canadian Conf. on Artificial Intelligence*, Montreal, May 1986.

Brown, C.M. (ed). *Advances in Computer Vision*. Vols. I and II. Hillsdale, NJ: Lawrence Erlbaum Associates, Pub., 1988.

Brown, C.M., "Color, texture, shape and motion," to appear, *Pattern Recognition Letters*, 1987.

Brown, C.M., "The Hough transform," in *Encyclopedia of Artificial Intelligence*, S. Shapiro (ed.), New York: John Wiley & Sons, Inc., in press, 1987.

Brown, C.M., "Space-Efficient Hough Transformation for Object Location," in *Statistical Image Processing and Graphics*. E. Wegman (ed). Marcel-Dekker, 1987.

Brown, C.M., J. Aloimonos, M. Swain, P. Chou, and A. Basu, "Texture, contour, shape and motion,"to appear, *Pattern Recognition Letters*, 5, 2, 1987.

Brown, C.M. and D.H. Ballard, "Vision: Biology challenges technology," invited article, *BYTE 10*, 44, 245-261, April 1985.

Brown, C.M., C.S. Ellis, J.A. Feldman, S.A. Friedberg, and T.J. LeBlanc, "Artificial intelligence research on the Butterfly multiprocessor," *Proc., Workshop on AI and Distributed Problem Solving*, National Academy of Sciences, Washington, DC, 109-118, May 1985.

Brown, C., R. Fowler, T. LeBlanc, M. Scott, M. Srinivas, L. Bukys, J. Costanzo, L. Crowl, P. Dibble, N. Gafter, B. Marsh, T. Olson, L. Sanchis, "DARPA parallel architecture benchmark study," Butterfly Project Report 13, Computer Science Dept., Univ. Rochester, October 1986.

Brown, C.M., T.J. Olson and L. Bukys, "Low-level image analysis on a MIMD architecture," submitted, Int'l. Conf. on Computer Vision, London, June 1987.

Bukys, L., "Connected component labeling and border following on the BBN Butterfly parallel processor," Butterfly Project Report 11, Computer Science Dept., Univ. Rochester, October 1986.

Bukys, L. and T.J. LeBlanc, "Getting started with the BBN Butterfly parallel processor," Butterfly Project Report 1, Second Edition, Computer Science Dept., Univ. Rochester, October 1986.

Chou, P.B. and R. Raman, "Experiments on stochastic relaxation on Markov random fields with multiple obervations," forthcoming technical report, Computer Science Dept., Univ. Rochester, 1987.

Chou, P.B. and D. Sher, "Multi-modal segmentation using Markov random fields," *Proceedings*, DARPA Image Understanding Workshop, Los Angeles, CA, February 1987.

Coombs, D.J. and C.M. Brown, "User's guide to the color analysis package at Rochester," internal publication, Computer Science Dept., Univ. Rochester, 1986.

Coombs, D.J. and B.D. Marsh, "Roving eyes -- Prototype of an active vision system," CSC400 project report, Computer Science Dept., Univ. Rochester, December 1986.

Cooper, P.R. and S.C. Hollbach, "Parallel recognition of objects comprised of pure structure," to appear, *Proceedings*, DARPA Image Understanding Workshop, Los Angeles, CA, February 1987.

Cooper, P.R., D.E. Friedman, and S.A. Wood, "The automatic generation of digital terrain models from satellite images by stereo," *36th Congress, Int. Astronautical Federation*, Stockholm; to appear, *Acta Astronautica, 1986.*

Costanzo, J., L. Crowl, L. Sanchis and M. Srinivas, "Subgraph isomorphism on the BBN Butterfly multiprocessor," Butterfly Project Report 14, Computer Science Dept., Univ. Rochester, October 1986.

Crowl, L., "Chrysalis++," Butterfly Project Report 15, Computer Science Dept., Univ. Rochester, December 1986.

Dutta, R., "Invariant symmetric complex movements," submitted to AAAI-87, Computer Science Dept., Univ. Rochester, 1986.

Ellis, C.S., "Concurrency and linear hashing system," *1985-86 Computer Science and Engineering Research Review*, Computer Science Dept., Univ. Rochester, September 1985.

Ellis, C.S., "Distributed data structures: A case study," *IEEE Trans. on Computers C-34*, 1178-1185, 1985.

Ellis, C.S. and R.A. Floyd, "Work in progress: Distributed and multiprocessor file systems," *SOSP 10*, December 1985.

Fanty, M.A., "A connectionist simulator for the BBN Butterfly multiprocessor," Butterfly Project Report 2, Computer Science Dept., Univ. Rochester, January 1986.

Feldman, J.A., "A functional model of vision and space," in *Vision, Brain and Cooperative Computation*. M. Arbib and A. Hanson (eds). Cambridge, MA: MIT Press/Bradford Books, in press, 1987.

Feldman, J.A., "Energy and the behavior of connectionist models," submitted to *Cognitive Science*, 1986.

Feldman, J.A., "Energy methods in connectionist modelling," to appear, *Pattern Recognition. Theory and Applications*, P.A. Devijver (ed.), NATO ASI Series in Computer Science, Springer Verlag, 1987.

Feldman, J.A., "Massively parallel computational models," in M. Commons (ed). Cambridge, MA: Harvard Univ, Press, in press, 1987.

Feldman, J.A., "Neural representation of concepts," submitted to *Science*, 1986.

Feldman, J.A., "Neural Representation of Conceptual Knowledge," to appear, forthcoming book, L. Nadel *et al.* (eds.), 1987; also, TR 189, Computer Science Dept., Univ. Rochester, June 1986.

Feldman, J.A., "Connectionist models and parallelism in high level vision, *CVGIP* **31**, Special Issue on Human and Machine Vision, 178-200, 1985.

Feldman, J.A., "Energy and the behavior of connectionist models," TR 155, Computer Science Dept., Univ. Rochester, November 1985.

Feldman, J.A., D.H. Ballard, C.M. Brown, and G.S. Dell, "Rochester connectionist papers: 1979-1985," TR 172, Computer Science Dept., Univ. Rochester, December 1985.

Feldman, J.A. and C.M. Brown, "Recent progress of the Rochester image understanding project," *Proc., DARPA Image Understanding Workshop*, Miami, FL, December 1985.

Feldman, J.A., J.L. McClelland, G. Bower and D. McDermott, "Connectionist models and cognitive science: Goals, directions and implications," position paper from NSF Workshop on Connectionist Modelling, 1986.

Feldman, J.A. and L. Shastri, "Evidential reasoning in semantic networks," to appear, forthcoming book, Wilks and Partridge (eds.), 1987.

Finkel, R.A., M.L. Scott, W.K. Kalsow, et al., "Experience with Charlotte: Simplicity vs. function in a distributed operating system," Computer Sciences TR 653, Univ. Wisconsin-Madison, July 1986; for *IEEE Workshop on Design Principles for Experimental Distributed Systems*, Purdue Univ., October 1986.

Fowler, R.J., "The complexity of using forwarding addresses for decentralized object finding," *Proceedings*, 5th Ann. ACM Symp. on Principles of Distributed Computing, Calgary Alberta, August 1986, 108-120.

Fowler, R.J., "A non-commutative logic of distributed protocols," submitted, 6th ACM Symp. on Principles of Distributed Computing, 1986.

Frederickson, G.N. and M.A. Srinivas, "On-line updating of solutions to a class of matroid intersection problems," to appear, *Information and Control.*

Friedberg, S.A., "A consistency protocol for highly available, replicated databases," submitted, *5th Symp. on Reliabiloity in Distributed Software and Database Systems*, 1986.

Friedberg, S.A., "Finding axes of skewed symmetry," *Computer Vision, Graphics, and Image Processing*, 34, 138-145, 1986.

Friedberg, S.A., "Hierarchical processor composition," *Proc., 14th ACM CS Conf.*, February 1986.

Friedberg, S.A., "HPC coding style guidelines," Hierarchical Process Composition Project Report 1, Computer Science Dept., Univ. Rochester, May 1986.

Friedberg, S.A., "Interface structures," Hierarchical Process Composition Project Report 5, Computer Science Dept., Univ. Rochester, August 1986.

Friedberg, S.A., "Symmetry evaluators" (revised), TR 134, Computer Science Dept., Univ. Rochester, January 1986.

Friedberg, S.A., "User process--HPC interface--C language/UNIX host version," Hierarchical Process Composition Project Report 3, Computer Science Dept., Univ. Rochester, August 1986.

Friedberg, S.A. and G.L. Peterson, "An efficient solution to the mutual exclusion problems using weak semaphores," submitted, *Information Processing Letters*, 1986.

Friedberg, S.A. and D.H. Pitcher, "HPC IPC implementation--unmodified UNIX host version," Hierarchical Process Composition Project Report 2, Computer Science Dept., Univ. Rochester, June 1986.

Friedberg, S.A. and I. Rigoutsos, "Comments on Stony Brook MP," Hierarchical Process Composition Project Report 4, Computer Science Dept., Univ. Rochester, May 1986.

Hinkelman, E., "NET: A utility for building regular process networks on the BBN Butterfly parallel processor," Butterfly Project Report 5, Computer Science Dept., Univ. Rochester, February 1986.

Hinkelman, E., "Pattern: A computational approach to quantifying coating quality," Eastman Kodak Research Labs Technical Report, to appear, 1986.

Hollbach, S.C., "Connectionist conceptual combination," internal document, Computer Science Dept., Univ. Rochester, December 1986.

Hollbach, S.C., "Tinker toy recognition from 2D connectivity," TR 196, Computer Science Dept., Univ. Rochester, October, 1986.

Kautz, H., "Formalizing Spatial Concepts and Spatial Language," in *Formal Theories of the Commonsense World*. J. Hobbs (ed). Stanford, CA: Center for the Study of Language and Information, 1985.

LeBlanc, T.J., "Problem decomposition and communication tradeoffs in a shared-memory multiprocessor," to appear, *Proceedings*, Workshop on Numerical Algorithms for Parallel Computer Architectures, Springer-Verlag, 1987.

LeBlanc, T.J., "Shared memory versus message-passing in a tightly-coupled multiprocessor: A case study," *Proc., 1986 Int. Conf. on Parallel Processing*, October 1986; Recipient of Distinguished Presentation Award. Also appears as Butterfly Project Report 3, Computer Science Dept., Univ. Rochester, revised February 1987.

LeBlanc, T.J. and R.P. Cook, "High-level broadcast communication for local area networks," *IEEE Software*, Special Issue on Experiences with Distributed Systems, 40-48, May 1985.

LeBlanc, T.J. and S.A. Friedberg," Hierarchical process composition in distributed operating systems," *Proc., 5th Int. Conf. on Distributed Computing Systems*, Denver, CO, 26-34, May 1985.

LeBlanc, T.J. and S.A. Friedberg, "HPC: A model of structure and change in distributed systems," *IEEE Trans. on Computers*, C-34, 12, 1114-1129, 1985; TR 153, Computer Science Dept., Univ. Rochester, May 1985.

LeBlanc, T.J., N.M. Gafter, and T. Ohkami, "SMP: a message-based programming environment for the BBN Butterfly," Butterfly Project Report 8, Computer Science Dept., Univ. Rochester, July 1986.

LeBlanc, T.J. and J.M. Mellor-Crummey, "Debugging parallel programs with instant replay," Butterfly Project Report 12 and TR 194, Computer Science Dept., Univ. Rochester, September 1986; to appear, *IEEE Trans. on Computers*, April 1987.

Litman, D.J., "Plan recognition and discourse analysis: an integrated approach for understanding dialogues," Ph.D. thesis and TR 170, Computer Science Dept., Univ. Rochester, September 1985.

Lynne, K.J., "Graphics interface to Rochester connectionist simulator," internal publication, Computer Science Dept., Univ. Rochester, 1986.

Marsh, B.D. and T.J. LeBlanc, "Prometheus: A model of computation for the BBN Butterfly," internal document, Computer Science Dept., Univ. Rochester, 1987.

Marsh, B.D. and T.J. LeBlanc, "Minimum cost path," *Proceedings*, DARPA Parallel Architecture Workshop, McLean, VA, October 1986.

Mellor-Crummey, J.M., "Reproducible execution of parallel programs," Research summary, Computer Science Dept., Univ. Rochester, Summer 1986.

Mellor-Crummey, J.M. and T.J. LeBlanc, "Instrumentation for distributed systems," Position paper submitted to ACM SIGARCH Workshop on Instrumentation for Distributed Systems, September 1986.

Narayanan, N.H. and N. Viswanadham, "A methodology for knowledge acquisition and reasoning in failure analysis of systems," *Proc., Symp. on AI in Engineering*, Washington, DC, October 1985; to appear, *IEEE Trans. on Systems, Man and Cybernetics*, 1986.

Newman-Wolfe, R.E., "Communication issues in parallel processing," Ph.D. thesis and TR 200, Computer Science Dept., Univ. Rochester, September 1986.

Newman-Wolfe, R.E., "A set theoretic problem: a solution for the hypercube and its applications," TR 171, Computer Science Dept., Univ. Rochester, October 1985.

Ohkami, T. and D. Baldwin, "The SEEDS simulator: User manual and report," internal document, Computer Science Dept., Univ. Rochester, 1986.

Olson, T.J., "An image processing package for the BBN Butterfly parallel processor," Butterfly Project Report 9, Computer Science Dept., Univ. Rochester, August 1986.

Olson, T.J., "Finding lines with the Hough transform on the BBN Butterfly parallel processor," Butterfly Project Report 10, Computer Science Dept., Univ. Rochester, August 1986.

Olson, T.J., "Modula-2 on the BBN Butterfly Multiprocessor", Butterfly Project Report 4, Computer Science Dept., Univ. Rochester, January 1986.

Quiroz, C., "Compilation for MIMD Architectures," thesis proposal, Computer Science Dept., Univ. Rochester, 1986.

Rigoutsos, I., "Homotopies: A panacea or just another method?" TR201, Computer Science Dept., Univ. Rochester, December 1986.

Rigoutsos, I., "Spatio-temporal energy models for the perception of motion: An implementation," Computer Science Dept., Univ. Rochester, December 1986.

Rigoutsos, I. and C.M. Brown, "Camera calibration," TR 186, Computer Science Dept., Univ. Rochester, to appear, 1986.

Sanchis, L.A., "Multiple-way network partitioning," TR 181, Computer Science Dept., Univ. Rochester, March 1986.

Scott, M.L., "Design and implementation of a distributed systems language," Ph.D. thesis, TR 596, Univ. Wisconsin-Madison, May 1985.

Scott, M.L., "The interface between distributed operating system and high-level programming language," Proc., *1986 Int. Conf. on Parallel Processing*, St. Charles, IL, August 1986, 242-249; also appears as TR 182 and Butterfly Project Report 6, Computer Science Dept., Univ. Rochester, January 1986; *Computer Science/Engineering Research Review*, Univ. Rochester, September 1986.

Scott, M.L., "Language support for loosely-coupled distributed programs," *IEEE Transactions on Software Engineering*, Special Issue on Distributed Computing, December 1986; TR 183, Computer Science Dept., Univ. Rochester, January 1986.

Scott, M.L., "LYNX reference manual," Butterfly Project Report 7, Computer Science Dept., Univ. Rochester, March 1986, revised August 1986.

Scott, M.L. and A.L. Cox, "An empirical study of message-passing overhead," submitted, 7th Int'l. Conf. on Distributed Computing Systems.

Scott, M.L. and R.A. Finkel, "A simple mechanism for type security across compilation units," *IEEE Transactions on Software Engineering*, Special Issue on Distributed Computing, to appear.

Shastri, L. and J.A. Feldman, "Neural Nets, Routines and Semantic Networks," in *Advances in Cognitive Science*. N. Sharkey (ed). Ellis Horwood Publishers, 1986.

Sher, D.B., "Advanced likelihood generators for boundary detection," TR197, Computer Science Dept., Univ. Rochester; submitted to Int'l. Conf. on Computer Vision, London, England, January 1987.

Sher, D.B., "Evidence combination based on likelihood generators," TR192, Computer Science Dept., Univ. Rochester; revised version submitted to IJCAI, January 1987.

Sher, D.B., "Appropriate and inappropriate estimation techniques," *Proceedings*, IEEE Workshop on Uncertainty and Artificial Intelligence, June 1986.

Sher, D.B., "Optimal likelihood generators for edge detection under Gaussian additive noise," *Proc., IEEE Conf. on Computer Vision and Pattern Recognition*, Miami, FL, June 1986; TR 185, Computer Science Dept., Univ. Rochester, June 1986; *1986-87 Computer Science and Engineering Research Review*, Computer Science Dept., Univ. Rochester, October 1986.

Sher, D.B., "Developing and analyzing boundary detection operators using probabilistic models," *Proc., ACM Workshop on Uncertainty and Probability in Artificial Intelligence*, August 1985.

Sher, D.B., "Evidence combination for vision, using likelihood generators," *Proc., DARPA Image Understanding Workshop*, Miami, FL, December 1985.

Sher, D.B., "Template matching on parallel architectures," TR 156, Computer Science Dept., Univ. Rochester, July 1985.

Swain, M.J. and J.L. Mundy, "Experiments in using a theorem prover to prove and develop geometrical theorems in computer vision," *1986 IEEE Int. Conf. on Robotics and Automation*, San Francisco, 280-285, April, 1986.

Swain, M.J., "Algorithms," *Queen's Mathematical Communicator*, Queen's College, to appear, 1986.


DAT
FILM
2